Convert submodule to builtin

March 2020

Contact Information

Name	Shourya Shukla
Major	Computer Science and Engineering
E-mail	shouryashukla.oo@gmail.com
IRC	rasengan_chidori on #git & #git-devel
Mobile no	+91 9871030887
GitHub	periperidip
Linkedin	shuklashourya
StackOverflow	rasengan__
Website	https://sites.google.com/view/periperidip
Address	119, Shree Awas Apartments, Sector-18B, Dwarka, New Delhi
Postal code	110078
Time Zone	IST (UTC +0530)

Background

I am Shourya Shukla, a sophomore in Computer Science and Engineering at the Indian Institute of Technology Roorkee. I was introduced to programming at a young age and I have been trying to learn new concepts everyday since. My interests include modern mobile networks, Internet of Things, system software development and cryptography. I had been an active member of InfoSec IITR, a group for information security enthusiasts. I have been working on a research project which involves providing cellular network access to users in a disaster-struck area via drones. I love low-level coding and FLOSS as well. I have been an active part of the Git community since January of this year, contributing to Git.

Work Environment

I am fluent in C/C++, Java and Shell script, and have an understanding of Python as well. I use Git as my VCS and Visual Studio Code with integrated GDB as my primary code editor and Ubuntu 19.10 as my primary Operating System unless the work specifically demands Windows.

Contributions to Git

Contributing to Git helped me understand a lot about how modern softwares work behind the walls as well as how real world development takes place. I will keep contributing to Git, be it solving doubts, reporting/solving bugs, writing code, etc. and make my work count. As of now, my contributions at Git are:

status: `merged`

git/git:

[Microproject]: `Modernise tests and use helper functions in test script.`

GitHub:https://github.com/git/git/commit/c513a958b69090c02ad422b0cd4622009bb4b9f5

List:https://lore.kernel.org/git/20200116203622.4694-1-shouryashukla.oo@gmail.com/

List

[Solved doubt]: `fatal: cannot rebase with locally recorded submodule modifications`

List:https://lore.kernel.org/git/20200207220403.28961-1-shouryashukla.oo@gmail.com/

List

[Aided a new contributor]: `Need help to start contributing`

List:https://lore.kernel.org/git/20200205032925.5272-1-shouryashukla.oo@gmail.com/

List

[Aided a potential GSoC student]: `[GSoC] Microproject for git`

List:https://lore.kernel.org/git/20200318192719.1127-1-shouryashukla.oo@gmail.com/

List

[Reviewed a Microproject]: `[GSoC][PATCH 1/2] t4131: modernize style`

List:https://lore.kernel.org/git/20200319163817.4239-1-shouryashukla.oo@gmail.com/

The Project: Convert `submodule` to builtin

Outline

Some Git commands were initially implemented directly in shell script with some instances of Perl as well. As times progressed, various platforms to run Git emerged & projects became large(spanning millions of lines of code), *enter*, problems in production level code:

- **There were issues with portability of code.** The submodule shell script uses commands such as `echo`, `grep`, `cd`, `test` and `printf` to name a few. When switching to non-POSIX compliant systems, one will have to re-implement these commands specifically for the system. There are also POSIX-to-Windows path conversion issues. To fix these issues, it was decided to convert these scripts into portable C code(the original intention C was developed with, *to have portable code and software*).
- **No access to low-level git API.** Shell commands don't have an access to the low-level git API, *i.e.*, the plumbing commands of git which include `git cat-file`, `git hash-object`, etc. These are the commands responsible for supporting the main git commands such as add, commit, etc. It is an obvious fact that the low level API will be needed to carry out any operation, but due to lack of access to it, the commands have to spawn a separate shell for the same.
- **There is large overhead involved in calling the command.** As these commands implemented in shell script are not buitlins, they tend to call multiple `fork()` and `exec()` syscalls for creating more child processes hence creating another shell. This is the aforementioned overhead we are talking about and it rather takes a huge toll on big repositories in terms of the time elapsed to run a command as well as the extra memory consumed.
- If commands have other nested commands (such as `git submodule` using `git rev-parse`, `git ls-files` and `git add` to name a few), the overhead mentioned in the point above tends to rise exponentially which again would contribute to the slowing down of the whole git suite. This again reinforces the above two reasons.

Various commands have been converted as of now due to the reasons mentioned above, such as add, blame, commit, bisect(work in progress), etc. In my project, I intend to **convert submodule into C fully**, hence making it a *'builtin'*.

Submodules and `git submodule`

Submodule, as defined in the gitglossary is, "*A repository that holds the history of a separate project inside another repository (the latter of which is called superproject).*", which translates to, an independent git repository inside another git repository.

Submodules are used when we need to use some work from an external repository(say we need a particular library*(eg: boost)* to implement in our code). Hence, we clone this repository as a submodule using `git submodule add <repo-url>`. The advantages are **three fold**:

- First, we can utilise the repository we just cloned, *i.e.*, the submodule in whatever way we wish. It would feel as if this submodule was just a part of the code. Hence, our use-case isn't damaged in any way whatsoever.
- Second, the history of this submodule is independent of the history of our working tree, *i.e.*, the superproject. This means that any commit we make in our submodule won't get rebased(*i.e.*, put on top) onto our superproject's working tree but would show up just as *any* other change in a directory of our working tree. Hence, our working tree's history isn't tampered.
- Finally, as a submodule is just a repository, we can conduct normal git actions such as pull, push, fetch, etc. on them. This helps in keeping our submodule up-to-date with the changes that are happening in it.

Git, for instance, uses the `sha1collisiondetection` repository as a submodule.

`git submodule` is a command used to manipulate and deal with submodules. Our aim is to convert this command from its shell form into its C form.

Previous Work

There has been ongoing work in the conversion of various Git commands such as `add`, `commit`, `blame`, etc. from their shell form into their C form. `git submodule` is one of the commands left to fully convert into its C form. Stefan Beller converted a large part of this command up until 2019. Prathamesh Chavan also aided in the conversion of the command during his GSoC project in the year 2017. In its current state, **four** `git submodule` subcommands are **due for conversion**, namely: `add`, `set-branch`, `set-url` and `summary`. Also, the Command Line parsing Interface needs improvements, such as better error messages and support for more subcommands.

Prathamesh implemented and improved the subcommands `status`, `sync`, `deinit` and some more. The relevancy of this to my project is that some helper functions(located in `submodule.c`) such as `print_submodule_summary()`,`prepare_submodule_summary()`, etc. have been

implemented beforehand. In the case of subcommand summary, the work left is to use these functions, integrate them with the basic scaffolding(*mentioned in the table below*) and implement the `module_summary()` frontend function. He also ported various helper functions such as `set_name_rev()`. He kept offering improvements to his conversions till around January of 2018.

Stefan Beller finished the implementation of the subcommand `init` as well as laid its foundation. He implemented `foreach` and improved `deinit` & `update` as well. He also ported various helper functions such as `resolve_relative_url()`.

Current Status of the subcommand and future vision

The **general format** of the BASH version for any subcommand is:

- A `cmd_subcommand-name()` function which **houses the main command functionality**. *eg:* `cmd_add()` for the add subcommand; `cmd_set_url()` for the set-url subcommand.
- Various other helper functions which **aid in the functioning of the whole command** and are used throughout the whole shell file by various subcommands and functions. *eg:* `get_submodule_config()`[helps in gathering the submodule configuration], `sanitize_submodule_env()`[helps in sanitizing the working tree by saving some git config parameters and clearing the local git environment].

The **general format** of the C version for any subcommand is:

- A `module_subcommand-name()` function which **acts as the frontend of the command** by accepting various parameters(such as `quiet`) and separating out the nuts and bolts of the command given then finally passing control on to other functions mentioned below. *eg:* `module_status()` for the status subcommand; `module_init()` for the init subcommand.
- A `subcommand-name_submodule()` function which is **generally the main working function**. Its job ranges from handling the parameters passed on by the aforementioned function to performing the main task of the subcommand. *eg:* `status_submodule()` for the status subcommand; `init_submodule()` for the init subcommand.
- A `subcommand-name_cb` structure which **contains variables** `prefix` **and** `flags` to contain the flags(parameters) and prefixes passed into the command line. *eg:* `struct status_cb{}` for the status subcommand.
- An **optional** callback function of the format `subcommand-name_submodule_cb()` which h**elps in performing callbacks**, *i.e.*, calls to other functions. *eg:* `status_submodule_cb()` for the status subcommand.
- A `SUBCOMMAND-NAME_CB_INIT` macro. *eg:* `INIT_CB_INIT` for the init subcommand.

- Various other helper functions which might be command specific(*eg:* `print_status()`) or might be useful for other subcommands and functions as well (*eg:* `get_default_remote()`).

There is also a `cmd_struct` called `commands`, which houses all the converted(*i.e.*, usable) commands along with some helper subcommands too. The structure will be updated when we port a command fully.

We will aim to port our **remaining subcommands: add, summary, set-branch and set-url** in the above mentioned format. The first job will be to create the frontend function, structure and macro(which constitutes the basic scaffolding) followed by creating the main working function with various helper functions on the way.

The current status of the conversion as well as the direction I will take for the conversion of the subcommands are as follows:

Subcommand	**Current status**
`add`	pending conversion, full code needs to be written for the same. Need to implement callback macros and structures, *i.e.* struct add_cb, ADD_CB_INIT, as well as frontend function module_add(). Other helper functions may be needed in the process as well. Compare with shell script and try to "translate" it into C. I guesstimate around 400-500 lines of code for this(including helper functions).
`set-branch`	pending conversion, full code needs to be written for the same. Need to implement macros and structures, *i.e.* struct setbranch, SETBRANCH_CB_INIT, as well as frontend function module_setbranch(). Other helper functions(*such as* `remote_submodule_branch()` *&* `get_default_remote()` *which are already implemented may prove helpful later*) may be needed in the process as well. Compare with shell script and try to "translate" it into C. This subcommand may take about 200 lines of C code to implement(including helper functions).
`set-url`	pending conversion, full code needs to be written for the same. Need to implement macros and structures, *i.e.* struct seturl, SETURL_CB_INIT, as well as frontend function module_seturl(). Other helper functions(*such as* `relative_url()` *&* `resolve_relative_url()` *which are already implemented may prove helpful later*) may be needed in the process as well. Compare with shell script and try to "translate" it into C. It will have a similar implementation to set-branch because they are "setter" functions. This subcommand may take about 200 lines of C code to implement(including helper functions).
`summary`	pending conversion, **work in progress**; callback structures, functions and macros have been created, also, basic scaffolding of the command is

	done, i.e., functions module_summary(), summary_submodule(), summary_submodule_cb(). As this is a prototype, some functions may be scrapped or added later. Other functions to complement the subcommand have already been created; learn from Prathamesh's mistakes and implement a better code. After discussions with Junio C Hamano, I intend to **add a "--recursive" option** as well for summary so as to obtain summaries of nested submodules as well. I estimate about 400 lines of code for this subcommand(excluding the **"--recursive" option**, yet including the helper functions)
`status`	conversion complete, currently in a functional state.
`init`	conversion complete, currently in a functional state.
`deinit`	conversion complete, currently in a functional state.
`update`	conversion complete, currently in a functional state.
`foreach`	conversion complete, currently in a functional state.
`sync`	conversion complete, currently in a functional state.
`absorbgitdirs`	conversion complete, currently in a functional state.

The commands which are in a converted state still use the shell script for accepting user input from the command line, followed by parsing the parameters and then forward this to the C file. To make submodule a **complete builtin**, this might need to be amended as well.

The subcommands set-url and set-branch might not require the complex wiring add and summary subcommands will need because of the fact that they are "setter" subcommands.

Though, there is about a 3 year gap between Prathamesh & Stefan's work and mine, the model for porting seems to be consistent even if coding style may vary and might even give out improvements over previous implementations.

Contribution process and interaction with the mentors

I will keep committing changes on my GitHub fork and finally post a patch series on the Mailing List. I will make sure to keep interacting with the community as well as the mentors regularly.

I aim to write weekly "progress report" blogs, which I will post on my website as well as the List. Apart from that, I will document anything new I learn as well as my journey in the GSoC program on my blogs and maybe as self-answered questions on StackOverflow with the aim that they will help me as well as others in case of reference.

Project Timeline

I have been studying the code of `submodule.c`, `submodule--helper.c` and `git-submodule.sh` since the submission of my microproject. After studying the codes, I tried to devise an effective conversion strategy for 'submodule'. I noticed that `submodule.c` contains various helper functions for `submodule--helper.c` whereas the latter houses the main "converted" command as of now.

The subcommands 'set-branch' and 'set-url' will provide easy conversion due to the vast array of helper functions already available for them. Hence, I intend to implement them before the other subcommands due to their simplicity in implementation as well as the motivation it will give me to do more.

After considering a lot of things, and important advice from Christian Couder, I have decided that I will **first implement** 'set-url' and 'set-branch', followed by 'summary' and finally 'add'. Integration testing and documentation updates will keep following the implementations. To add on, the conversion of summary might become a tad bit easier due to the existence of a patch to convert it, which will aid me in learning from the mistakes committed before and thus help me offer an even more improved version of the subcommand. .

Therefore, after all these considerations, the timeline looks like:

- Empty Period (Present - May 4)

➜ I am writing a paper**(on the project I have been working upon)** for a conference which I have to finalise and submit by the first week of April. Hence, I might be inactive in that period.

➜ My end-semester exams begin on April 23(tentative, *may* change due to the Corona pandemic) hence I might be a bit busy a week or so before their commencement as well as the 14 days in which exams take place.

➜ I plan on starting the conversion of 'set-url' and 'set-branch' in this period. Although I might be a little occupied I will try my best to implement a basic scaffolding and maybe even complete some good portions of the subcommands and will keep my mentors posted regarding the same.

- Community Bonding Period (May 5 - June 1)

➜ Get familiar with the community

➜ Improve the project workflow: make some timeline changes if necessary.

➜ Finish implementation of 'set-url' and 'set-branch' subcommands

➜ Update the Documentation

- Phase 1 (June 2 - July 3)

➜ Convert ‘summary’ subcommand

➜ Improve CLI parsing(give out better error messages)

➜ Update the Documentation

➜ Add appropriate tests for integration testing of ‘set-url’, ‘set-branch’ and ‘summary’

- Phase 2 (July 4 - August 10)

➜ Convert ‘add’ subcommand

➜ Improve the remaining bits of the CLI parsing

➜ Update the Documentation

➜ Add appropriate tests for integration testing of ‘add’ with the whole system

- Final Phase (August 11 - August 24)

➜ Improve and add Documentation(*if* there is any still left)

➜ Apply final touch-ups to code

If there is some extra time left, I will try to implement some **BONUS** features.

BONUS features: Consist of command touch ups and improving some bugs such as code sections with *'NEEDSWORK'* tags, improving the test files and maybe improve some previous implementations of helper functions. Also, there are some incomplete bits of the ‘update’ subcommand as well in the shell file, as pointed out by Dscho, which may need to be corrected.

Workflow

I have divided the project into 3 subprojects(SP).

1. **SP 1:** Convert ‘set-branch’ and set-url’
2. **SP 2:** Convert ‘summary’ and and improve CLI(Command Line Interface) parsing
3. **SP 3:** Convert ‘add’ and improve CLI parsing

After discussions with Christian Couder, I plan to start SP1 before GSoC itself. Currently, I am studying the code in detail and constructing a scaffolding for this implementation. I aim to complete the leftover bits(in any) of SP1 during Phase 1 and SP2 & SP3 during Phase 2 of GSoC.

As Derrick Stolee advised, the conversion may not be possible in one whole summer, hence, I think an early start might be needed to finish things in time if possible.

As of now***(March 21(UTC))***, my progress is described by the following commit. I have implemented the frontend function(*almost*) `module_summary()`. I hope to increase my work speed once I get a hang of the inner working and coding style of the command.

Availability

The official GSoC period starts from April 27 and ends on August 17. My vacations start from May 10 and will be over by July 13. I can easily devote 45-50 hours per week until the commencement of my Semester. Other than this project, I have no commitments planned for my vacations. I shall keep the community posted in case of any change in plans.

Post GSoC

Even after the completion of Google Summer of Code, I plan on continuing my contributions to Git, on the technical front(in terms of code and documentation contributions) as well as on the social front(solving people's doubts/problems on the List as well as on StackOverflow). I vision to convert the remaining of the commands as pointed out by Dscho as well as improve the test files.

I aim to develop mentorship skills as well as the ability to guide others and try to give back to the community by mentoring and guiding others as well(by reviewing their code, helping them out, etc.)

Final Remarks

I have a habit of not giving up. I will keep trying things until I succeed at them. Same was my case with learning to use Git in my freshman year. I was so scared of it for some reason that I refrained from using ‘git bash’. But I knew that I had to master this tool(or at least learn it to a *satisfactory* extent) because of the utility it has in a programmer’s life. I kept going, watching tons of tutorials, reading the documentation and articles and Lo, here I am writing code for Git.

I hope that you give me the chance to showcase my abilities by considering my proposal for working with you during the summer of 2020. I will try my best to keep the bar high and deliver good work. Really looking forward to learning from you :)

Kind Regards,

Shourya Shukla